

GO-03 01: Модули и пакеты. Пакеты в Go

Описание:

Весь код на Go состоит из пакетов (package). Пакеты позволяют удобно организовывать зависимости и переиспользовать код в приложении. Существует два типа пакетов executable и reusable:

- executable - исполняемые пакеты. Это всегда пакет main, который содержит в себе точку входа в приложение - обязательную функцию main(). Исполняемые пакеты не могут быть импортированы в другой пакет.
- reusable - это пакеты библиотек. Предназначены для выделения и переиспользования частей кода в разных местах приложения или вовсе в других приложениях. Переиспользуемые (reusable) пакеты не могут быть самостоятельно скомпилированы и запущены. Пакет состоит из одного или нескольких файлов, находящихся в одной директории и имеющих одно значение директивы package.

Давайте создадим приложение Hello World в базовом окружении Go, в директории gopackages и рассмотрим это приложение в контексте пакетов.

```
└─ $GOPATH
   └─ src
      └─ gopackages
         └─ main.go
```

файл main.go
`package main`

`import "fmt"`

```
func main() {
    fmt.Println("Hello world")
}
```

Все файлы в .go должны начинаться с объявления пакета директивой package. Так определяется, к какому пакету относится тот или иной .go файл. В данном случае наш пакет называется main и имеет функцию main(), что делает его исполняемым пакетом (executable).

Директива import подключает reusable пакет - fmt. После его подключения мы можем вызывать функции, описанные в этом пакете. Например, далее мы вызываем функцию fmt.Println(), которая выводит сообщение на экран.

Пробуем запустить нашу программу.

```
$ go run main.go
Hello world
```

Импорт стандартных пакетов

В этом примере мы использовали сторонний код из пакета `fmt` в нашей программе. Получилось это благодаря тому, что мы импортировали в наш пакет `main` библиотеку `fmt` из стандартной поставки Go. То есть разработчики языка создали эту библиотеку и положили ее в коробку с самим Go. Поэтому при импорте мы указали только название нужной библиотеки и получили ее без каких-либо дополнительных действий. В поставке Go есть еще много полезных библиотек. Вот их полный список: <https://golang.org/pkg/>.

Импорт сторонних пакетов

Импорт происходит очень просто, когда мы используем стандартные пакеты Go типа `fmt`. Но что, если нам необходимо использовать библиотеку, созданную другими разработчиками? Как мы знаем, это очень частая необходимость при разработке приложений. И на этот случай в Go есть специальная утилита `go get` (и еще много других способов, но о них не в этом задании), которая позволяет получить Go-библиотеку из любого источника, например, из `github.com`. А потом импортировать ее в наш пакет. В прошлом примере мы выводили текст функцией `fmt.Println()`. Тогда текст выводился белым цветом. Теперь предположим, что нам нужно вывести текст красным. Для этого есть хорошая библиотека на гитхабе <https://github.com/fatih/color>. Давайте попробуем ее использовать. Для начала нам нужно скачать библиотеку в рабочее окружение Go. Делается это следующим образом:

```
$ go get github.com/fatih/color
```

После выполнения этой команды в папку `$GOPATH/src/` качается весь код библиотеки со всеми ее зависимостями.

```
└─ $GOPATH
    └─ src
        └─ github.com
            └─ fatih
                └─ color
                    └─ color.go
                    └─ ...
            └─ gopackages
                └─ main.go
```

И теперь мы можем сделать импорт этой библиотеки в наше приложение через директиву `import`.

Изменим код:

```
package main
```

```
import "fmt"
import "github.com/fatih/color"

func main() {
    fmt.Println("Hello world")
    color.Red("Hello world again")
}
```

Запускаем:

```
$ go run main.go
Hello world
Hello world again
```

В консоли получаем текст, вывод которого в одном случае произошел с помощью стандартного пакета `fmt`. А в другом - с помощью библиотеки, написанной сторонним разработчиком, которую мы нашли на `github`. Обратите внимание, что в Go принято называть директорию с пакетом также, как и сам пакет. Поэтому путь к пакету выглядит так - `github.com/fatih/color`.

Создание собственных пакетов и их импорт

В предыдущих примерах мы рассмотрели импорты стандартных пакетов Go и пакетов, написанных сторонними разработчиками. Теперь же приступим к написанию собственных локальных `reusable` пакетов, которые позволят нам структурировать код приложения и сделать его переиспользуемым там, где это нужно.

Создадим пакет `wordz`, который будет уметь генерировать случайные слова. Для этого в нашем проекте добавим еще одну директорию, а внутри нее - `go` файл. Назовем их `wordz`. В Go принято использовать одно и то же название для директории пакета и для основного файла в пакете.

```
└─ $GOPATH
   └─ src
      └─ github.com
         └─ fatih
            └─ color
               └─ color.go
                  ...
      └─ gopackages
         └─ main.go
            └─ wordz
               └─ wordz.go
```

Файл `wordz.go`

```

package wordz

import "crypto/rand"
import "math/big"

var Hello = "This is package wordz"
var Prefix = "Random word: "
var Words = []string{"One", "Two", "Three", "Four", "Five"}

func Random() string {
    max := len(Words)
    r, _ := rand.Int(rand.Reader, big.NewInt(int64(max)))
    return get(r.Int64())
}

func get(index int64) string {
    return Prefix + Words[index]
}

```

Рассмотрим код пакета wordz подробнее.

package wordz - это название пакета. Если название пакета не main, значит, такой пакет считается reusable пакетом. И его не получится запустить и использовать без импорта.

Если мы прямо сейчас попытаемся запустить код этого пакета, то получим ошибку:

```

$ go run wordz/wordz.go
go run: cannot run non-main package

```

import "math/rand" - импортируем стандартный пакет math/rand. Он нужен нам для генерации случайных чисел. Напоминаю, что при импорте Go сначала смотрит в \$GOROOT/src/, где лежат все стандартные пакеты, а затем - в \$GOPATH/src/, куда скачиваются пакеты с помощью утилиты go get. В данном случае импортируемый пакет лежит по пути \$GOROOT/src/math/rand/.

Объявляем три переменные.

```

var Hello = "This is wordz package"
var Prefix = "Random word: "
var Words = []string{"One", "Two", "Three", "Four", "Five"}

```

Переменная Hello содержит приветственное сообщение. Переменная Prefix - это префикс для более красивого вывода. А переменная Words - это набор слов, из которых мы будем выбирать случайное.

Дальше две функции `Random` и `get`. `Random` - основная функция нашего пакета, которая, собственно, и будет возвращать нам случайное слово. Она генерирует случайное число и передает его на вход в другую функцию `get`, которая возвращает слово из слайса `Words` по переданному числу, используя его в качестве индекса.

Экспортируемый и неэкспортируемый код

Если посмотреть внимательно на наш пакет, то мы увидим, что какие-то переменные и функции называются с большой буквы, а какие-то - с маленькой. Это вовсе не случайно. Потому что таким образом в Go определяется область видимости элементов на уровне пакетов. Элементы (переменные, функции, структуры и т.д.), названные с большой буквы, являются экспортируемыми элементами (`exported`) и могут быть использованы вне пакета. А названные с маленькой буквы - неэкспортируемые (`unexported`), могут использоваться только в том пакете, в котором они определены. Это похоже на модификаторы доступа `public` и `private` из других языков (например `php`, `java` или `python`). В нашем пакете мы основную функцию `Random` и переменную `Hello` назвали с большой буквы, определив их как экспортируемые.

Итак, в целом наш пакет готов. Давайте попробуем использовать его в программе. Для этого в файл `main.go` необходимо добавить еще одну директиву импорта `import "gopackages/wordz"` Импорт в данном случае сработает, используя переменную `$GOPATH` (`$GOPATH/src/gopackages/wordz`)

```
└─ $GOPATH
   └─ src
      └─ github.com
         └─ fatih
            └─ color
               └─ color.go
                  └─ ...
      └─ gopackages
         └─ main.go
            └─ wordz
               └─ wordz.go
```

После импорта можно использовать функцию `Random` и переменную `Hello`, вызвав их по имени пакета.

```
wordz>Hello
wordz.Random()
```

Выведем их значения через `fmt.Println` в цикле.

```
for i := 0; i < 5; i++ {
    fmt.Println(wordz>Hello)
    fmt.Println(wordz.Random())
}
```

Файл main.go

```
package main

import "fmt"
import "github.com/fatih/color"
import "gopackages/wordz"

func main() {
    fmt.Println("Hello world")
    color.Red("Hello world again")

    for i := 0; i < 5; i++ {
        fmt.Println(wordz.Hello)
        fmt.Println(wordz.Random())
    }
}
```

```
$ go run main.go
Hello world
Hello world again
This is wordz package
Random word: Two
This is wordz package
Random word: Three
This is wordz package
Random word: Three
This is wordz package
Random word: Five
This is wordz package
Random word: Two
```

Отлично! Все получилось. В пакете main был использован созданный нами пакет wordz. С помощью него мы сгенерировали и вывели несколько случайных слов.

Функция init()

Теперь мы рассмотрим еще одну возможность пакетов в Go - это функция init(). Она по сути своей является аналогом конструктора из других языков. init() вызывается один раз при инициализации пакета. Давайте посмотрим, как это работает.

Добавим следующие блоки в файл wordz.go

```
import "fmt"
```

```
func init() {  
    fmt.Println("Function init in package wordz")  
}
```

Файл wordz.go

```
package wordz
```

```
import "crypto/rand"  
import "math/big"  
import "fmt" //Не забываем импорт пакета fmt
```

```
var Hello = "This is wordz package"  
var Prefix = "Random word: "  
var Words = []string{"One", "Two", "Three", "Four", "Five"}
```

```
func init() { // Добавили функцию. Она сработает при импорте пакета  
wordz в файле main.go  
    fmt.Println("Function init in package wordz")  
}
```

```
func Random() string {  
    max := len(Words)  
    r, _ := rand.Int(rand.Reader, big.NewInt(int64(max)))  
    return get(r.Int64())  
}
```

```
func get(index int64) string {  
    return Prefix + Words[index]  
}
```

```
$ go run main.go  
Function init in package wordz  
Hello world  
Hello world again  
This is wordz package  
Random word: Two  
This is wordz package  
Random word: Three  
This is wordz package  
Random word: Three
```

```
This is wordz package
Random word: Five
This is wordz package
Random word: Two
```

Не меняя ничего в файле main.go, запускаем программу и видим, что первый вывод у нас произошел из функции init() пакета wordz (Function init in package wordz). Это означает, что запуск функции происходит в момент исполнения директивы import "gopackages/wordz". То есть, при инициализации пакета. Функцию init() можно использовать, когда нужно, например, запустить программу с определенной конфигурацией параметров или в определенном состоянии.

Виды импорта и синтаксис

В файле main.go мы импортировали три разных пакета, используя такой синтаксис:

```
import "fmt"
import "github.com/fatih/color"
import "gopackages/wordz"
```

Вызывали директиву импорт три раза. Но в Go также есть более красивый способ это сделать:

```
import (
    "fmt"
    "github.com/fatih/color"
    "gopackages/wordz"
)
```

Такая запись смотрится лаконичнее, особенно при условии, что в файл могут импортироваться много пакетов.

Виды импорта

У импорта в Go есть еще несколько видов. Виды импорта решают проблемы, которые могут возникать при работе с пакетами. Так выглядит синтаксис разных видов импорта:

```
import (
    "fmt" //обычный импорт
    somename "some/package/one" //импорт с использованием псевдонима для
пакета
    _ "some/package/two" //импорт через underscore
    . "some/package/three" //импорт через точку
)
```

Начнем рассматривать по порядку.

Обычный импорт

Импортируем пакет "fmt". С этим видом мы уже хорошо знакомы. Он просто импортирует пакет в нужный нам файл, и мы по названию пакета можем обращаться к его элементам. В нашем случае - `fmt.Println("Hello world")`.

Импорт с использованием псевдонима для пакета

`somename "some/package/one"`

В первую очередь, такой импорт необходим для избежания конфликтов в названиях импортируемых пакетов, если нужно импортировать два разных пакета, но эти пакеты имеют одинаковое название. В нашей программе есть пакет `github.com/fatih/color`, для вызова его функций мы обращались по названию `color` `color.Red("Hello world again")`. Но что будет, если мы создадим собственный пакет с названием `color` и попробуем его импортировать?

```
└─ $GOPATH
    └─ src
        └─ github.com
            └─ fatih
                └─ color
                    └─ color.go
                    └─ ...
        └─ gopackages
            └─ main.go
            └─ wordz
                └─ wordz.go
            └─ color
                └─ color.go
```

файл `src/gopackages/color/color.go`

```
package color
```

```
import "fmt"
```

```
func Greet() {
    fmt.Println("This is new package Color")
}
```

Импортируем его в нашу программу, пробуем запустить функцию `Greet`.

`"gopackages/color"`. Импорт в данном случае сработает, используя переменную `$GOPATH` (`$GOPATH/src/gopackages/color`).

файл `main.go`

```
package main
```

```
import (
```

```

    "fmt"
    "github.com/fatih/color"
    "gopackages/wordz"
    "gopackages/color" // Добавили импорт локального пакета
)

func main() {
    color.Greet() //Вызываем функцию с приветствием
    fmt.Println("Hello world")
    color.Red("Hello world again")

    for i := 0; i < 5; i++ {
        fmt.Println(wordz.Hello)
        fmt.Println(wordz.Random())
    }
}

$ go run main.go
# command-line-arguments
./main.go:8:2: color redeclared as imported package name
    previous declaration at ./main.go:6:2
./main.go:13:2: undefined:
    "_/Users/dhnikolas/go/src/gopackages/color".Red

```

После запуска мы видим, что Go ругается на то, что пакет с таким именем уже был проимпортирован. Исправить эту ситуацию нам помогут псевдонимы при импорте. Вместо обычного импорта мы присвоим нашему новому локальному пакету псевдоним newcolor "gopackages/color" и дальше для вызова функции Greet будем использовать не имя пакета color, а его псевдоним newcolor.Greet()

```
package main
```

```

import (
    "fmt"
    "github.com/fatih/color"
    "gopackages/wordz"
    newcolor "gopackages/color" // Добавили импорт локального пакета
    через псевдоним
)

func main() {

```

```
newcolor.Greet() //Вызываем функцию из локального пакета color по  
его псевдониму
```

```
fmt.Println("Hello world")  
color.Red("Hello world again")
```

```
for i := 0; i < 5; i++ {  
    fmt.Println(wordz.Hello)  
    fmt.Println(wordz.Random())  
}  
}
```

```
$ go run main.go  
Function init in package wordz  
This is new package Color  
Hello world  
Hello world again  
This is wordz package  
Random word: Two  
This is wordz package  
Random word: Three  
This is wordz package  
Random word: Three  
This is wordz package  
Random word: Five  
This is wordz package  
Random word: Two
```

Теперь все работает правильно. Мы вызвали функции из обоих пакетов color.

```
newcolor.Greet()  
color.Red("Hello world again")
```

В одном случае - по псевдониму, а в другом - по оригинальному названию пакета.

Импорт через underscore

```
_ "some/package/two"
```

Сначала поймем проблему, которую решает этот импорт. В Go пакеты, импортированные в файл, должны быть обязательно использованы. Иначе будет паника. Давайте в файле main.go уберем вызовы пакета wordz и запустим программу.

файл main.go

```
package main
```

```
import (
```

```

    "fmt"
    "github.com/fatih/color"
    "gopackages/wordz"
    newcolor "gopackages/color"
)

func main() {
    newcolor.Greet()
    fmt.Println("Hello world")
    color.Red("Hello world again")

    //Удалили фрагмент кода с вызовами пакета wordz
}

```

```

$ go run main.go
# command-line-arguments
./main.go:6:2: imported and not used:
"/Users/dhnikolas/go/src/gopackages/wordz"

```

Получаем панику. Go говорит нам, что пакет wordz импортирован, но не используется. Так вот, импорт с нижним подчеркиванием позволит нам импортировать пакет wordz, вызвать в нем функцию `init()`, но при этом мы сможем не вызывать и не обращаться к элементам пакета явно. Добавляем underscore для импорта wordz. `_ "gopackages/wordz"` файл main.go

```

package main

import (
    "fmt"
    "github.com/fatih/color"
    _ "gopackages/wordz" //Добавляем underscore для импорта
    newcolor "gopackages/color"
)

func main() {
    newcolor.Greet()
    fmt.Println("Hello world")
    color.Red("Hello world again")
}

```

```
$ go run main.go
Function init in package wordz
This is new package Color
Hello world
Hello world again
```

Теперь программа работает. И обратите внимание, что произошел вызов функции `init()` в пакете `wordz` - `Function init in package wordz`.

Импорт с точкой

```
. "some/package/three"
```

Импорт с точкой добавляет все экспортируемые поля пакета в область видимости файла.

И теперь мы можем работать с полями импортированного пакета так, как будто они у нас в пакете. Рассмотрим на примере пакета `wordz`. Добавим его на этот раз через точку. .

`"gopackages/wordz"` И обращаемся к переменной `Hello` и функции `Random` прямо из пакета `main`, не обращаясь к ним по имени пакета.

```
package main
```

```
import (
    "fmt"
    "github.com/fatih/color"
    . "gopackages/wordz" //Добавляем пакет wordz через точку
    newcolor "gopackages/color"
)
```

```
func main() {
    newcolor.Greet()
    fmt.Println("Hello world")
    color.Red("Hello world again")

    fmt.Println>Hello) //Вызов переменной из пакета wordz
    fmt.Println(Random())//Вызов функции из пакета wordz
}
```

```
$ go run main.go
Function init in package wordz
This is new package Color
Hello world
Hello world again
This is wordz package
Random word: Two
```

Все работает так, как будто переменная `Hello` и функция `Random` объявлены в пакете `main`. Нужно сказать, что использование импорта с точкой - это совсем не частая практика. И скорее исключение. Этот способ импорта не рекомендуется использовать из-за его неясности. И сложно представить ситуацию, в которой такой импорт был бы жизненно необходим :)

На этом мы заканчиваем тему пакетов и подводим итог!

- Пакеты в Go бывают двух видов - `executable` и `reusable`.
- Директива `package` - для объявления пакета, `import` - для подключения пакета.
- Пакеты могут импортироваться из разных источников - стандартные пакеты библиотеки Go, пакеты из удаленных репозиториях типа `github` и локальные пакеты.
- Бывают экспортируемые (`exported`) элементы пакета, которые начинаются с заглавной буквы, и неэкспортируемые (`unexported`) - начинаются с маленькой.
- Функция `init()` выполняется при инициализации пакета, в том числе при импорте.
- Директива импорт имеет четыре вида - обычный, через псевдоним, через нижнее подчеркивание и через точку.

Задание:

1. Форкните репозиторий [module03](#) с кодом данного задания - в группу с вашими репозиториями - `golang_users_repos/<your_gitlab_id>`.
2. Создайте у себя в проекте `module03` из ветки `master` ветку `01_task`.
3. Создайте в проекте `module03` пакет (с произвольным названием), который будет состоять из двух `go` файлов и содержать 2 функции, по одной в каждом файле:
 - `City()` - возвращает случайный город,
 - `Digit()` - возвращает случайное число строчного типа (`one`, `two`, `three` и т.д.).
 - Обе функции должны формировать результат с помощью функции `Random` из пакета `wordz`. При этом не внося никаких изменений в пакет `wordz`.
 - Выведите через `fmt.Println` результат функции `City` и `Digit` в файле `main.go`
4. С помощью утилиты `go get`, установите пакет для работы со строками [xstrings](#).
5. В файле `main.go` примените функцию `Shuffle` из этого пакета к результату функции `City()`. И угадайте, какое название города вывелось :)
6. В ответе пришлите ссылку на `merge request` в ветку `master` своего проекта ветки `01_task`.